Fault-Criticality Assessment for AI Accelerators using Graph Convolutional Networks

Arjun Chaudhuri†, Jonti Talukdar†, Jinwook Jung‡, Gi-Joon Nam‡, and Krishnendu Chakrabarty†
†Department of Electrical and Computer Engineering, Duke University, Durham, NC
‡IBM Thomas J. Watson Research Center, Yorktown Heights, NY

***Abstract*—Owing to the inherent fault tolerance of deep neural networks (DNNs), many structural faults in DNN accelerators tend to be functionally benign. In order to identify functionally critical faults, we analyze the functional impact of stuck-at faults in the processing elements of a 128×128 systolic-array accelerator that performs inferencing on the MNIST dataset. We present a 2-tier machine-learning framework that leverages graph convolutional networks (GCNs) for quick assessment of the functional criticality of structural faults. We describe a computationally efficient methodology for data sampling and feature engineering to train the GCN-based framework. The proposed framework achieves up to 90% classification accuracy with negligible misclassification of critical faults.**

I. Introduction

Advances in deep neural networks (DNNs) are driving the demand for domain-specific accelerators (DSAs) [1]. Training of machine learning (ML) models is typically carried out in cloud datacenters [2]. On the other hand, inferencing can be carried out using systolic array-based energy-efficient DSAs on edge devices [3]. A systolic array is a two-dimensional array consisting of identical processing elements (PEs) that performs multiply-and-accumulate (MAC) operations.

To ensure safety in applications such as autonomous driving, built-in self-test (BIST) is used to detect in-field failures in AI accelerators [4], [5]. However, many structural faults in the PE array do not have a significant impact on inferencing accuracy [6]. Hence, generating BIST-based pseudo-random test patterns to target all faults in the hardware is an “over-kill”.

In this paper, we analyze the impact of structural faults in the PE array on inferencing accuracy and present an efficient technique to assess the inferencing error in the presence of a fault. The main contributions of this paper are as follows:

- We present an ML-based classification technique that uses graph convolutional networks (GCNs) to evaluate the functional criticality of structural faults in a PE for a given DNN-to-accelerator mapping and inferencing workload.
- We describe the systematic selection of ground-truth data and features for training and validation of the GCN models.
- We present the design-space exploration (DSE) of a 2-tier GCN-based framework to reduce misclassifications.

The remainder of the paper is organized as follows. Section II reviews systolic array-based accelerators and highlights the challenges associated with functional criticality analysis. Section III describes GCN training with appropriate feature engineering. Section IV presents the ground-truth selection technique and the 2-tier GCN-based framework. Section V compares the effectiveness of GCNs with that of DNNs for criticality classification, and evaluates the 2-tier GCN-based framework. Finally, Section VI concludes the paper.

II. Background and Motivation

A. AI Accelerators and Fault Injection Framework

Systolic-array accelerators offer high throughput, high compute density, and improved performance per Watt for MAC operations. We designed a 128×128 systolic array, supporting 16-bit and 32-bit IEEE FP data formats, in Verilog HDL. The array was synthesized using Nangate 45 nm cell library. Each PE in the array includes an adder and multiplier. This design is based on state-of-the-art accelerator architectures such as Google's TPU [2]. We used the LeNet-5 DNN trained on the MNIST dataset as our target application [7].

We developed a structural fault-injection framework for injecting stuck-at (s-a-0, s-a-1) faults, delay faults and bridging faults in the gate-level netlist of the PE. The criticality of a fault is assessed on the basis of its impact on the inferencing accuracy over a representative test set of 100 MNIST images. While the framework is general, the analysis presented in this work is limited to the impact of single stuck-at faults.

B. Efficient Criticality Assessment of Structural Faults

The inherent fault-tolerance of DNNs can be attributed to the robustness of training on large datasets, non-linearity of activation units, and the use of regularization features such as dropout [6], [8]. Furthermore, the application of DNNs to different use-cases provides flexibility in terms of acceptable performance thresholds required to ensure functional correctness. This flexibility can be leveraged to develop metrics to grade structural faults in terms of their functional criticality.

The *functional criticality* of a fault is determined by the *severity* of its impact. For inferencing, this impact is quantified in terms of the loss in inferencing accuracy for the given use-case. A structural fault is considered to be *functionally critical* if the inferencing accuracy under the presence of the fault is below the *criticality threshold*, A_{min}, determined by the safety-criticality of the application. A fault that is not functionally critical is deemed to be *benign*.

Functional fault-criticality assessment can be used to guide test effort and quality assessment of AI accelerators throughout the product life-cycle. As shown in Fig. 1, it can be applied to various domain-specific use-cases, which include specific DNN models mapped to the accelerator hardware for different applications. As functional fault-criticality depends on the type of dataset, criticality threshold of the application, and model-to-hardware mapping, fault criticality can be cataloged for domain-specific use-cases.

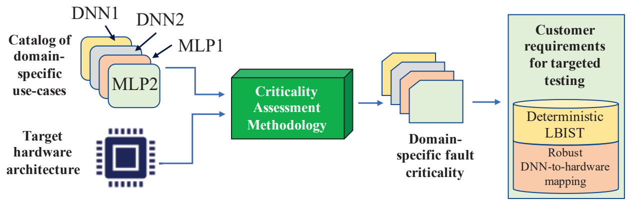
DNN1
DNN2
MLP1
MLP2
Catalog of domain-specific use-cases
Target hardware architecture
Criticality Assessment Methodology
Domain-specific fault criticality
Customer requirements for targeted testing
Deterministic LBIST
Robust DNN-to-hardware mapping

Fig. 1: Fault-criticality assessment for domain-specific use-cases.

Pattern generation for logic BIST (LBIST) during in-field testing is driven by ATPG-generated seeds stored in on-chip memory [9]. ATPG-based reseeding can be directed to target random-pattern-resistant critical faults across all catalogs, helping in their early detection, and reducing pattern count and test-time for in-field tests. Early detection of critical faults will prevent catastrophic failure of safety-critical systems and enabling timely product recall. Moreover, given a domain-specific use-case, efficient fault-criticality assessment enables DSE to determine robust DNN-to-accelerator mappings.

Fault criticality can be assessed by performing functional simulations for structural faults. However, brute-force functional simulation is prohibitively expensive due to high CPU runtime. This motivates the need for a scalable and efficient method to evaluate the functional criticality of structural faults.

C. *Limitations of Related Prior Work*

Prior work to analyze the impact of soft errors on DNN accelerators does not evaluate the criticality of structural faults [6]. Recent work [10] on functional testing of DNN architectures relies on random sampling of structural faults and does not take hardware reuse into account. Other recent work on functional testing of DNNs mapped to systolic-array accelerators considers randomly injected faults only at the PE pin-level, providing no insight into the criticality of faults within the PE [11].

III. GCN FOR CRITICALITY EVALUATION

A. *GCN Architecture*

A GCN is an ML model based on semi-supervised learning; it leverages the topology of a graph for classification of nodes in the graph [12]. The gate-level netlist of the PE can be represented as a directed graph G, where the nodes represent gates and edges represent interconnections. If both s-a-0 and s-a-1 faults at a node output are functionally benign, the node is labeled as functionally benign; otherwise, it is labeled as critical. Unlike a DNN that uses features of a stand-alone node to learn or predict its criticality, the forward-propagation rule in GCN uses feature information of a node and its neighbors to justify or evaluate its criticality. In recent years, GCNs have been applied to testability analysis and transistor sizing in mixed-signal circuits [13], [14].

To train a DNN for node-criticality classification, the topological features of a node X must be hand-extracted and explicitly provided by the user; it is often uncertain whether a set of topological features sufficiently describes a node and can guarantee convergence in training. In contrast, GCN implements feature aggregation of neighboring nodes to classify

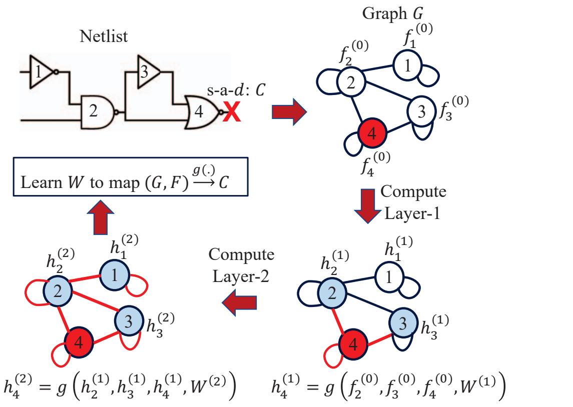
Netlist
Graph G
s-a-d: C
$f_1^{(0)}$
$f_2^{(0)}$
$f_3^{(0)}$
$f_4^{(0)}$
Learn W to map $(G,F)\xrightarrow{g(.)}C$
Compute Layer-1
Compute Layer-2
$h_1^{(1)}$
$h_2^{(1)}$
$h_3^{(1)}$
$h_1^{(2)}$
$h_2^{(2)}$
$h_3^{(2)}$
$h_4^{(2)} = g\left(h_2^{(1)}, h_3^{(1)}, h_4^{(1)}, W^{(2)}\right)$
$h_4^{(1)} = g\left(f_2^{(0)}, f_3^{(0)}, f_4^{(0)}, W^{(1)}\right)$

Fig. 2: Netlist-to-graph conversion and layer-wise forward-propagation during the training of a 2-layer GCN. The blue nodes contribute to feature aggregation in a red node via the red edges.

the criticality of X. Therefore, it captures the intricate node embeddings in G and does not need topological features to be provided explicitly.

The GCN architecture is similar to that of a feedforward fully-connected classifier. For the training and evaluation of a GCN, the netlist-graph G is saved as an undirected graph with a symmetric adjacency matrix A to allow: (i) bi-directional transfer of feature information between adjacent nodes; (ii) feature aggregation of a node and its neighbors. A feature matrix $F^{(0)}$ contains the user-defined feature vectors of all nodes in G and has dimensions $n \times f$; here n is the number of nodes in G and f is the number of features describing each node in G. During layer-wise forward propagation with L layers, normalized feature aggregation in the l-th layer is expressed as: $F^{(l)} = D^{-1} \cdot A \cdot H^{(l-1)}$, where $H^{(l-1)}$ is the output of $(l-1)$-th layer, D is the diagonal node-degree matrix, A is the adjacency matrix, and $F^{(l)}$ is the aggregated feature matrix which is an input to a non-linear transformation function $g(\cdot)$. Aggregation averages the feature vectors of a node and its neighboring nodes. Each node's features are updated with the corresponding aggregated features and are transformed to lower-dimensional representations using $g(\cdot)$. The output $H^{(l)}$ of the l-th layer is: $H^{(l)} = g(F^{(l)} \cdot W^{(l)})$, where $W^{(l)}$ is the weight matrix of the l-th layer. To enforce feature-dimensionality reduction, the number of columns in $W^{(l)}$ is set to be less than the number of columns in $F^{(l)}$. In [12], state-of-the-art classification results are obtained via symmetric normalization of the feature aggregation. Accordingly, the aggregation expression for $F^{(l)}$ is re-written as follows:

$$F^{(l)} = D^{-\frac{1}{2}} \cdot A \cdot D^{-\frac{1}{2}} \cdot H^{(l-1)}$$

Fig. 2 illustrates forward propagation during the training of a 2-layer GCN. The user-defined feature vector of the node numbered i ($i \in \{1,2,3,4\}$) is denoted by $f_i^{(0)}$. After the computation of layer l, the row-vector of $H^{(l)}$ corresponding to the node numbered i is denoted by $h_i^{(l)}$. The objective of training is to learn an optimal mapping of the features of a node (here, node numbered 4) and its neighborhood to the node label C. The features of all nodes that are at most L hops away from a given node are available for aggregation and transformation during the training and evaluation of an L-layer GCN. Thus, more layers in the GCN architecture implies that the feature information of a larger neighborhood is used to learn and evaluate node criticality.

B. *Feature Engineering*

Training of ML models for criticality classification requires topology-based, dataflow-based, and functional features. The topology-based features for a node X in the gate-level netlist include its depth from the nearest primary inputs and primary outputs, the number of PIs (POs) in its fan-in (fan-out) cone, and the number of gates in its fan-in and fan-out cones. The functional features include: (i) number of sign, exponent, and mantissa pins in the fan-out cone of X, (ii) gate type of X, (iii) histogram of different gate-types in the fan-in and fan-out cones of X, and (iv) probability of X's output being 0.

Dataflow-based features consist of individual image scores generated from fault-free bit streams through the node across all images in the test set. Each bit stream contains N_{cyc} bits, where N_{cyc} is the simulation cycle count for inferencing. The total number of bit streams equals the total number of test images in the test dataset. Each bit stream is compressed in a weighted fashion across the simulation cycles N_{cyc}, such that each cycle is uniquely represented in the final image score. If $b_{i,j}$ is the i^{th} bit of the j^{th} bit stream, then the dataflow score for that bit stream is $S_j = \sum_{i=1}^{N_{cyc}} b_{i,j} \cdot i$.

IV. 2-TIER GCN-BASED FRAMEWORK

A. *Node Sampling for Ground-Truth Collection*

To obtain a ground-truth set S_{GT} for training, we sample nodes from a directed netlist-graph G based on an user-provided radius of coverage (ROC), R_C. For traversing G with V nodes, the nodes in G are first ordered using a function $Arrange(G)$. If G contains cycles, $Arrange(G)$ performs a breadth-first-search on G; otherwise, $Arrange(G)$ performs a topological sort. The nodes are visited in the arranged order: if a newly visited node V_j is a root node with no incoming edges, it is added to S_{GT}. If the shortest distance D between V_j and a node in S_{GT} exceeds R_C, V_j is added to S_{GT}. Therefore, if a node is selected for ground-truth collection, all nodes lying within the ROC of the selected node are not included in S_{GT}. The worst-case time complexity of the proposed algorithm is $\mathcal{O}(V+E)$, where E is the number of edges in G. To label a node X in S_{GT}, functional fault simulation is performed on the representative dataset of an application (e.g., MNIST) to obtain the criticality of faults in X based on A_{min}.

B. *2-tier GCN-based Framework*

Training and Validation: The first tier applies a GCN model, referred to as GCN-1, to classify the criticality of a node. The labeled set of nodes S_{GT} is randomly split into training and validation sets; r_{tr} is the fraction of nodes in S_{GT} assigned to the training set. The adjacency matrix of G, functional and dataflow-based features of all nodes in G, and the criticality labels of the nodes in the training set are used to train GCN-1. The GCN-1 model is a feedforward fully-connected network

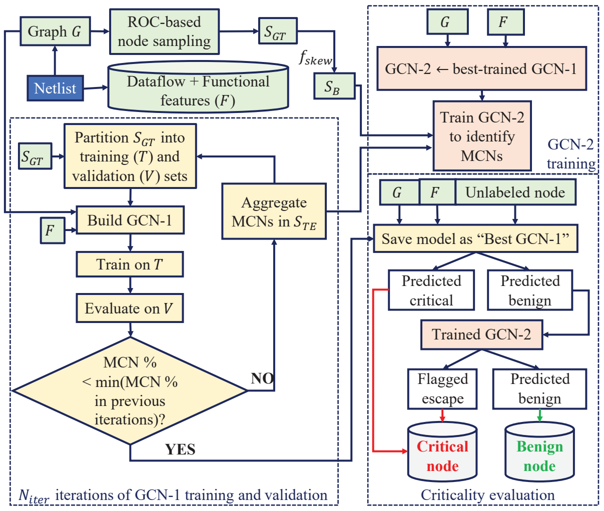
Graph G
ROC-based node sampling
S_{GT}
f_{skew}
S_B
Netlist
Dataflow + Functional features (F)
G
F
GCN-2 ← best-trained GCN-1
Train GCN-2 to identify MCNs
GCN-2 training
S_{GT}
Partition S_{GT} into training (T) and validation (V) sets
Aggregate MCNs in S_{TE}
F
Build GCN-1
Train on T
Evaluate on V
MCN % < min(MCN % in previous iterations)?
NO
YES
N_{iter} iterations of GCN-1 training and validation
G
F
Unlabeled node
Save model as "Best GCN-1"
Predicted critical
Predicted benign
Trained GCN-2
Flagged escape
Predicted benign
Critical node
Benign node
Criticality evaluation

Fig. 3: Training and evaluation of a 2-tier GCN-based framework.

with N_l layers. The input layer has I neurons, where I is the dimensionality of a node's features, and the output layer has two neurons for binary classification. The trained GCN-1 is validated on the nodes in the validation set. During validation, the GCN-1 may misclassify some critical nodes as benign. At the same time, some benign nodes may be misclassified as critical; such a scenario is considered to be a *false alarm*. We follow a conservative approach by prioritizing the minimization of critical-fault misclassification (CFM).

To reduce CFMs, the second tier uses another GCN model, referred to as GCN-2. The objective of GCN-2 is to learn the feature distribution of the critical nodes misclassified by GCN-1 and distinguish them from the benign nodes. With this objective, the weights of the best-trained GCN-1 model are re-trained to generate the weights of GCN-2. The architecture of GCN-2 is identical to that of GCN-1. The misclassified critical nodes (MCNs) obtained during the validation of GCN-1 are added to a set, S_{TE}. An identical number of benign nodes are selected from S_{GT} and added to a set, S_B. The nodes in S_{TE} and S_B are used to train GCN-2 to distinguish between an actual benign node and a critical node that has been misclassified as benign by GCN-1.

If the trained GCN-1 performs well on the validation set, the number of nodes in S_{TE} is low and may not be sufficient for training GCN-2. Therefore, the size of S_{TE} depends on the nodes in the training and validation sets, as well as on r_{tr} which determines the amount of training data for GCN-1. To aggregate more misclassification data for training GCN-2, we run N_{iter} ($N_{iter} > 1$) iterations of training and validation of GCN-1. For each iteration, the nodes in S_{GT} are randomly split into training and validation sets based on r_{tr}. The union of MCNs obtained after validation of GCN-1 across N_{iter} iterations constitutes S_{TE}. The GCN-1 version producing the least number of MCNs during validation across all the iterations is saved as the *best-trained* GCN-1 model.

The aggregation of misclassification data prioritizes GCN-2 training to reduce CFM. To limit the number of false alarms, the size of S_B is kept higher than that of S_{TE} to introduce a partial bias in GCN-2 towards benign classification. Hence, $n_B = \lceil f_{skew} \cdot n_{TE} \rceil$, where n_B and n_{TE} are sizes of S_{GT} and S_{GT}, respectively; f_{skew} is the skew factor ($f_{skew} > 1$).

Evaluation: During evaluation of the functional criticality of the unlabeled nodes in G, the adjacency matrix of G and the functional and dataflow-based features of all nodes in G are fed as inputs to the best-trained GCN-1 model. The nodes classified as benign by GCN-1 are then evaluated by the trained GCN-2 model for the potential detection of MCNs. If a node is classified as critical by either GCN-1 or GCN-2, it is considered to be functionally critical. Otherwise, it is considered to be functionally benign; see Fig. 3.

V. Experimental Results

A. Experimental Setup

The 2-tier GCN-based framework is implemented using an open-source PyTorch-based platform called Deep Graph Library (DGL) [15]. Feature engineering and node sampling are executed using a Python script in order to enable seamless integration with DGL-based implementation of GCNs. We have experimentally found that a 10-layer GCN model is sufficient for achieving high classification accuracies with zero CFM. The fault-free inferencing accuracies of 16-bit and 32-bit accelerators on the input dataset are 95% and 99%, respectively. To obtain node-criticality labels, we conservatively set A_{min} to 90% and 95% for 16-bit and 32-bit PEs, respectively, so that sufficient critical nodes are available for training. The framework is trained, validated, and evaluated on PE(20,0), i.e., the PE in the 21st row and first column of the systolic array.

B. Comparison between DNN and GCN

Table I presents a comparison between DNNs and GCNs for criticality evaluation. The model architectures, hyperparameters, and datasets are kept identical for DNN-based and GCN-based evaluation. The classification accuracy A_c is the percentage of nodes in the evaluation set whose criticality are predicted correctly by a trained ML model. Faults that cause less than 80% inferencing accuracy are considered as *catastrophically critical* faults. The catastrophically critical faults misclassified as benign are considered as CFMs; the percentage of such faults (out of the total number of critical faults in the evaluation set) is quantified by M_c. We compare between: (i) DNN provided with topological features (TF), (ii) DNN not provided with TF, and (iii) GCN not provided with TF. GCN-based criticality evaluation leads to smaller M_c compared to DNN-based evaluation. It automatically captures the topological embeddings of nodes in the netlist-graph for classification, thereby reducing the effort associated with manual feature extraction. Admittedly, M_c is non-negligible; we address this problem next using multi-tier GCNs.

C. DSE of 2-tier GCN-based Framework

In the DSE of a 2-tier GCN-based framework, a *configuration* of the framework refers to a four-element tuple $(L, N_1, f_{skew}, r_{tr})$. In our DSE experiments for 2-tier framework: $L \in \{7, 10\}$, $3 \le N_1 \le 7$, $f_{skew} \in \{2, 3, 4, 5\}$, and $r_{tr} \in \{0.6, 0.75\}$. Table II presents the results for the 32-bit adder and multiplier of PE(20,0), and for 16-bit PE(20,0). For each netlist, the configuration achieving the lowest M_c is shown. The 2-tier framework achieves $M_c \sim 0\%$. The percentage reduction in the number of faults to be targeted for in-field testing is denoted by Δ_S. Here, $\Delta_S = 100 \cdot N_B/N_T$, where N_B is the number of faults classified as benign and N_T is the total number of faults in the netlist. The results show a significant reduction in the number of structural faults to be targeted for in-field testing.

TABLE I: Comparison between DNN and GCN based evaluation.

Netlist	DNN with TF		DNN without TF		GCN without TF	
	A_c (%)	M_c (%)	A_c (%)	M_c (%)	A_c (%)	M_c (%)
Add32	94.7	7.9	80.9	42.7	73.4	1.2
Mult32	93.3	36.2	83.8	68.1	90.7	14.5
PE16	87.6	9.5	72.8	20.8	87.7	8.1

Add32 (Mult32): adder (multiplier) in 32-bit PE; PE16: 16-bit PE.

TABLE II: Performance summary of 2-tier GCN-based framework.

Netlist	Configuration	A_c (%)	M_c (%)	N_T	N_B	Δ_S (%)
Add32	(7,4,2,0.6)	81.2	1.2	6952	4562	65.6
Mult32	(7,5,2,0.6)	84.4	0	6525	5737	87.9
PE16	(10,4,2,0.6)	78.8	0	6415	2477	38.6

VI. Conclusion

We have presented a 2-tier GCN framework to analyze the functional criticality of structural faults in systolic-array accelerators. We have also presented an efficient technique for carefully selecting nodes and their features for training the criticality assessment framework.

Acknowledgement

This work was supported in part by the Semiconductor Research Corporation under GRC Task 2879.001.

References

[1] Y. Chen et al., "A survey of accelerator architectures for deep neural networks," *Elsevier Engineering*, 2020.
[2] N. P. Jouppi et al., "In-datacenter performance analysis of a tensor processing unit," in *ISCA*, 2017, pp. 1–12.
[3] "Google Edge TPU: Coral AI". https://coral.ai
[4] S. Shibahara, "Functional safety SoC for autonomous driving," in *CICC*, 2018.
[5] Y. Liu et al., "Deterministic stellar BIST for automotive ICs," *IEEE Trans. CAD*, 2019, doi:10.1109/TCAD.2019.2925353.
[6] G. Li et al., "Understanding error propagation in deep learning neural network (DNN) accelerators and applications," in *ACM SC*, 2017.
[7] Y. LeCun et al., "Gradient-based learning applied to document recognition," *Proc. IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998.
[8] B. Reagen et al., "Ares: A framework for quantifying the resilience of deep neural networks," in *DAC*, 2018.
[9] D. Xiang et al., "Low-power scan-based built-in self-test based on weighted pseudorandom test pattern generation and reseeding," *TVLSI*, 2016.
[10] A. Gebregiorgis et al., "Testing of neuromorphic circuits: structural vs functional," in *ITC*, 2019.
[11] J. J. Zhang et al., "Fault-tolerant systolic array based accelerators for deep neural network execution," *IEEE Design & Test*, 2019.
[12] T. N. Kipf et al., "Semi-supervised classification with graph convolutional networks," in *ICLR*, 2017.
[13] Y. Ma et al., "High performance graph convolutional networks with applications in testability analysis," in *DAC*, 2019.
[14] H. Wang et al., "TTS: transferable transistor sizing with graph neural networks and reinforcement learning," in *DAC*, 2020.
[15] Deep Graph Library (Python package). https://docs.dgl.ai/en/0.4.x/.